Implementing a Category-Theoretic Framework for Typed Abstract Syntax

Ahrens, Benedikt; Matthes, Ralph; Mörtberg, Anders

DOI
10.1145/3497775.3503678

Publication date
2022

Document Version
Final published version

Published in
CPP 2022

Implementing a Category-Theoretic Framework for Typed Abstract Syntax

Benedikt Ahrens
B.P.Ahrens@tudelft.nl
Delft University of Technology
Netherlands
University of Birmingham
United Kingdom

Ralph Matthes
ralph.matthes@irit.fr
IRIT, Université de Toulouse, CNRS,
Toulouse INP, UT3
Toulouse, France

Anders Mörtberg
anders.mortberg@math.su.se
Department of Mathematics,
Stockholm University
Stockholm, Sweden

Abstract

In previous work ("From signatures to monads in UniMath"), we described a category-theoretic construction of abstract syntax from a signature, mechanized in the UniMath library based on the Coq proof assistant.

In the present work, we describe what was necessary to generalize that work to account for simply-typed languages. First, some definitions had to be generalized to account for the natural appearance of non-endofunctors in the simply-typed case. As it turns out, in many cases our mechanized results carried over to the generalized definitions without any code change. Second, an existing mechanized library on ω-cocontinuous functors had to be extended by constructions and theorems necessary for constructing multi-sorted syntax. Third, the theoretical framework for the semantical signatures had to be generalized from a monoidal to a bicategorical setting, again to account for non-endofunctors arising in the typed case. This uses actions of endofunctors on functors with given source, and the corresponding notion of strong functors between actions, all formalized in UniMath using a recently developed library of bicategory theory. We explain what needed to be done to plug all of these ingredients together, modularly.

The main result of our work is a general construction that, when fed with a signature for a simply-typed language, returns an implementation of that language together with suitable boilerplate code, in particular, a certified monadic substitution operation.

CCS Concepts: • **Theory of computation → Algebraic semantics**; **Categorical semantics**; **Logic and verification**; *Type theory*.

CPP '22, January 17–18, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9182-5/22/01.
https://doi.org/10.1145/3497775.3503678

Keywords: typed abstract syntax, monad, signature, formalization, computer-checked proof.

ACM Reference Format:
Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. 2022. Implementing a Category-Theoretic Framework for Typed Abstract Syntax. In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '22), January 17–18, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3497775.3503678

1 Introduction

There are many frameworks for the specification and analysis of abstract syntax. Frequently, these frameworks are first developed for untyped syntax (see, e. g., Fiore et al. [29], Gabbay and Pitts [30], Hofmann [38]), with extensions to the typed case often omitted, or promised as future work (see, e. g., Ahrens et al. [8]). However, in practice, such extensions are not trivial (see, e. g., Fiore [25], Miculan and Scagnetto [48]). In the present paper, we report on the extension of one such framework—by Ahrens et al. [13]—from untyped to simply-typed syntax. That work uses a notion of signature for untyped languages in terms of *strong functors*; using Mendler iteration [47], it builds the syntax generated by any such signature, together with a monadic substitution operation. The work is mechanized in the proof assistant Coq [59], using the UniMath library of univalent mathematics [63].

In the untyped case, an abstract signature is given by an endofunctor $H : [C, C] \to [C, C]$, for a suitably structured category C (e. g., Set). For instance, the functor underlying the signature of the untyped λ-calculus is

$$H\ F = \mathsf{Id}_C + (F \times F) + (F \cdot \mathsf{option}) \quad .$$

We write $\cdot$ for functor composition in applicative order and functor application with a space, so $(F \cdot \mathsf{option})\ X = F\ (X + 1)$, which hence corresponds to F with an additional free variable that will be bound by the abstraction constructor. With the aim of initial algebra semantics the ω-cocontinuity of H is established and an initial algebra for H is constructed using a classical theorem of Adámek [1]. Such an initial algebra is an endofunctor $\Lambda : [C, C]$ with a(n iso)morphism $\alpha : H\ \Lambda \to \Lambda$ encoding the constructors. The fact that this is an initial algebra furthermore ensures that Λ has a suitable

induction principle, making it equivalent to the following, more familiar, inductive type:

```
Inductive LC (X : Type) : Type :=
  | Var : X → LC X
  | App : LC X → LC X → LC X
  | Lam : LC (X + 1) → LC X.
```

By equipping H also with a *strength*, specifying how substitution should act "homomorphically" on the constructors, the functor Λ may be equipped with a substitution operator following Matthes and Uustalu [46] (formalized in UniMath by Ahrens and Matthes [12]). This substitution satisfies the laws of a monad—an observation originating in the work of Altenkirch and Reus [18], Bellegarde and Hook [19], Bird and Paterson [20]. Thus the framework of [13] not only constructs datatypes as initial algebras, but also equips them with a certified—in the sense of satisfying formally verified laws—substitution operation. To ease the use of the framework, the strong functor H is constructed algorithmically from a simple notion of signature—resulting in a form of datatype-generic programming in UniMath.

In the present work, we extend this framework by both generalizing to the simply-typed case, as well as refining and analyzing many of the involved notions. Starting from a simple notion of multi-sorted signature (Section 2.1), we construct (Section 2.2) a corresponding signature functor $H : [C, C] \to [C, C]$, for some suitably structured category C (e. g., $\mathrm{Set}^{\mathsf{S}}$ for a (discrete, small) category S of sorts). This construction involves functors that are not endo, so the notion of strength had to be generalized to $H : [C, C] \to [C, \mathcal{D}]$—indeed, it can easily be generalized to $[C, \mathcal{D}'] \to [C, \mathcal{D}]$, for categories $\mathcal{D}$ and $\mathcal{D}'$. This generalization is purely formal and the old UniMath proofs worked without major changes. We find this quite remarkable and a good example of how the initial investment of mechanizing mathematics pays off in the long run—it would have been quite time-consuming to establish sufficient conditions on $\mathcal{D}$ and $\mathcal{D}'$ by hand, and then verify that the proofs still go through. On the way to this more general notion of abstract signature we also decompose the notion of signature with strength of [46] to clarify what is needed for the substitution operation (Section 2.3) and for it to satisfy the monad laws (Section 2.4).

In order to construct the abstract datatype specified by H as an initial algebra, as well as the substitution monad, we prove that H is ω-cocontinuous. Many of the results already in UniMath apply, but some were missing, in particular results about when post-composing with a functor is ω-cocontinuous (Section 3.1). Having done this, the results of [13] apply and we obtain the substitution operation as a monad (Section 3.2). Throughout the paper we rely on the simply-typed λ-calculus (STLC) as a running example, but we have also implemented more complex languages, including PCF (Example 3.9) and the pre-syntax of the calculus of constructions (Example 3.10). In the case of the STLC the obtained signature functor H is a bit more complex than in the untyped case (see Example 2.10), but the generated syntax is equivalent to the following inductive type:

```
Inductive STLC (X : S → Type) : S → Type :=
  | Var : ∀ t, X t → STLC X t
  | App : ∀ s t, STLC X (s ⇒ t) → STLC X s → STLC X t
  | Lam : ∀ s t, STLC (X + { s }) t → STLC X (s ⇒ t).
```

Here, S is a small set of sorts closed under $\Rightarrow$, and X is a set of sorted variables indexed by S. Furthermore, $X + \{ s \}$ denotes X with an additional variable of sort s.

Having successfully generalized the formalization in [13] to simply-typed signatures, we give in Section 4 a bicategorical analysis of the notion of strength of Definition 2.15. To this end we relate it to an action-based notion of strength as put forward by Pareigis [49], which also occurs in the work of Fiore [26]. This is then also related to a notion of relative strength of [12]. This section makes use of displayed categories [11] and of a library of bicategory theory [6] recently added to UniMath.

1.1 Contributions

We present a fully formalized mathematical framework for simply-typed syntax relying on a categorical construction of inductive families as initial algebras. In this way, we avoid meta-programming and complex nested inductive types, and instead work completely internally to UniMath. We generically equip the constructed syntax with a monadic substitution operation. Furthermore, we provide a bicategorical analysis of the involved notions, motivated by the generalization of strength needed to encompass multi-sorted languages.

All results in this paper have been formalized in UniMath. The code is integrated into the UniMath library [63]. For documentation, we refer to a specific version of UniMath, available in commit c26d11bef0b4d4c226ca854d8de340d684c2a10a. The HTML documentation derived from this version is hosted online. To connect the paper and the formalization we provide clickable identifiers (e. g., MultiSortedSig) for all definitions and results. By clicking on one of the links, one gets taken to the corresponding definition in the HTML documentation.

1.2 Related Work

There are many theoretical frameworks suitable for the study of abstract syntax, see, e.g., the work of Altenkirch et al. [17], Chapman et al. [22], Fiore [25, 26, 27], Fiore and Hamana [28], Gambino and Hyland [31], Hamana [32], Hamana and Fiore [33], Hirschowitz et al. [34], Miculan and Scagnetto [48], Tanaka and Power [58]. We are interested in the following properties of such a framework:

1. Is there a computer-checked implementation?
2. Is there an internal notion of signature?
3. Does it support multi-sorted languages?
4. What is the theory that it is implemented in?

1.2.1 Implementations. In this section, we only consider frameworks that satisfy property 1. We classify them into three categories according to property 2.

Into the first category fall frameworks that work with an *external* notion of signature. Such frameworks take a signature specified in some domain-specific language, e. g., as a `Haskell` data type, and return a library of computer code for some proof checker such as `Coq`.[1] To this category belong the tools `DBGen` [51], `Ott` [55], `LNGen` [64], `Autosubst` [54] and `Autosubst2` [56]. These are different from our approach in that they typically rely on some form of meta-programming as the notion of signature is external.

The second category consists of work building a library of generic functions and reasoning principles for (typed) abstract syntax. The library `NomPa` [52, 53] falls into this category, as does `GMETA` [41]. In this line of work, there are several modules called signatures, providing an interface for signatures. However, there is not a fixed mathematical definition of a signature, making them less suited for general mathematical study of syntax and more focused on convenience for specifying and working with a specific language.

Into the third category fall tools that have an *internal* definition of signatures, stated within the proof assistant. In this category, signatures are themselves first-class objects and may be reasoned about within the system. Our work falls into this category, and we thus spend some more time comparing it to other work in this category.

Ahrens [4] considers only untyped syntax. The work is fully implemented in the computer proof assistant `Coq`, and relies on `Coq`'s inductive types and associated structural recursion for the construction of syntax and substitution.

Work by Ahrens et al. [7, 9] only discusses untyped syntax, but construct, like us, syntactic models without relying on inductive types. Ahrens and Zsidó [14] consider a simple notion of signature akin to our multi-sorted signatures of Definition 2.1. To any such signature, they associate a category of models and construct an initial such model. For this construction, crucially, they rely on suitable mutually inductive type families. Compared to their work, we study here a more general notion of abstract signatures (Definition 2.15) together with a map from multi-sorted signatures to these abstract signatures. We then construct the initial models without relying on general inductive types, but rather on a category-theoretic construction of initial algebras. A set-theoretic construction of initial models closer to ours is given in the PhD thesis of Zsidó [65], but is not computer-checked.

Allais et al. [15, 16] consider "descriptions" (see also [23]) for signatures. Like Ahrens and Zsidó [14], they assume suitable inductive datatypes; descriptions then straightforwardly generate relative "term" monads. These monads should be initial in a suitable category, even though such categorical formulations are not used or explored in that work. In both works, a well-behaved monadic substitution operation is constructed via the induction mechanism of `Agda`. The framework of Löh and Magalhães [42] is suitable for specifying abstract syntax with binding; typed syntax is not discussed. Again, the construction of syntax relies on `Agda`'s datatypes.

In [24], the authors study two notions of signature similar to the ones studied here—multi-sorted binding signatures and signatures with strength—and also give a translation from the former to the latter, similar to our translation in Section 2. Like [14] they use inductive datatypes to construct syntax. Their notion of syntax includes "meta-variables".

In summary, compared to the aforementioned work, we study here the construction of syntax, and of a suitable substitution operation for syntax, from a *category-theoretic* rather than a *type-theoretic* perspective. Specifically, we construct syntax exploiting ω-cocontinuity of the signature functor, and substitution for that syntax using Mendler-style recursion, expressed categorically.

1.2.2 Comparison to Other Frameworks. Here, we review categorical frameworks for abstract syntax that are not, to our knowledge, mechanized.

Fiore, Plotkin, and Turi [29] consider untyped syntax specified by an untyped variant of binding signatures (Definition 2.1). The authors consider both a categorical formulation of recursion to specify substitution (on "free" objects), as well as a type-theoretic formulation using inductive types and structural recursion. However, no link is made between the category-theoretic and the type-theoretic formulations. The mathematical structure given by substitution and its properties is taken to be a "substitution monoid".

Fiore [26] considers a notion of signature (given by an endofunctor $\Sigma : C \to C$) with strength, like we do in the present work. They discuss the categorical construction of syntax and substitution as the colimit of an ω-cocontinuous functor, see [26, Cor. 4]. We discuss their notion of signature in more detail in Section 4. Fiore and Hamana [28] later consider polymorphically typed terms, such as System F. They do not discuss the construction of abstract syntax. Hur [39] considers, like we do here, category-theoretic constructions of syntax. The focus of that work is on equations between terms. Mahmoud [44] considers structural induction in a type-theoretic style; a category-theoretic construction of syntax or substitution is not discussed.

Work by Hirschowitz and Maggesi [36, 37] considers syntax and substitution in form of monads on the category of sets. Hirschowitz and Maggesi also implement a special case of their work—for the λ-calculus—in the computer proof assistant `Coq`. (A comparison between Fiore, Plotkin, and Turi's approach [29] using substitution monoids and Hirschowitz and Maggesi's using monads is given in Zsidó's thesis [65].) The first part of the present work, in particular, of Section 3, can be considered to be "zooming in" on Hirschowitz and

[1] This approach is called "generative" by Lee et al. [41].

Maggesi's work, notably on the construction of the initial monad, expressed purely categorically.

Ahrens [2, 3, 5] considers simply-typed signatures and the construction of the generated syntax in a type-theoretic setting. Some instances of the framework are formalized in Coq, but general signatures are not.

Hirschowitz, Hirschowitz, and Lafont [34] consider a very abstract notion of signature. They construct abstract syntax via categorical machinery, and furthermore the passage to a "quotient syntax" modulo a system of equations. In the extended version [35, Sec. 5.5], the authors explain the construction of a signature in their sense from a signature à la Fiore, Plotkin, and Turi [29].

1.3 UniMath and Category Theory Therein

In this section we provide a brief introduction to UniMath and fix notations used throughout the paper. We write $a = b$ for the type of identifications/equalities/paths from a to b, and $a \equiv b$ for definitional/judgmental equality—in particular, we write $a :\equiv b$ for defining a to be b. We do not rely on any inductive types other than the ones specified in the prelude of UniMath, such as identity types, sum types, natural numbers, and booleans. Indeed, part of the work consists of the construction of certain initial algebras from dependent products, dependent sums, identities, and natural numbers. We use the notions of propositions and sets of Univalent Foundations: a type X is a proposition if $\prod_{(x,y:X)} x = y$ is inhabited, and a set if the type $x = y$ is a proposition for all $x, y : X$. Hence, despite working in Coq, we do not rely on the universes Prop or Set.

However, our main constructions and results, while conveniently expressed using univalent foundations, are not dependent on the full univalence axiom. On top of Martin-Löf type theory [45], they rely on propositional and functional extensionality, and propositional resizing. The key consequence of these that we use is the construction of effective set quotients due to Voevodsky [61]. These axioms are compatible with the Uniqueness of Identity Proofs (UIP) principle, and hence with the interpretation of types as sets.

We assume there to be at least two type-theoretic universes, $\mathsf{U}_0 : \mathsf{U}_1$. We call a type $X : \mathsf{U}_0$ a *small* type. Following the convention established in the HoTT book [60], we leave the universe levels implicit throughout the paper.

A *category* C in UniMath is given by a type of objects C_0 and a family of sets $C(x, y)$ for any $x, y : C_0$. We call C small if objects and morphisms are types from U_0. We assume the reader to be familiar with the concepts of category theory as found in the standard text by Mac Lane [43]. We sometimes remark that specific categories are univalent [10]. Such results typically depend on the full univalence axiom, but none of our main results depend on categories being univalent. The reader who prefers to read our results outside of univalent foundations may hence safely ignore these remarks.

The category $\mathsf{Set} :\equiv \mathsf{Set}(\mathsf{U}_0)$ of sets has, as objects, sets from the universe U_0; this is a category whose types live in U_1, hence it is not small. The set quotients discussed above are used to ensure that Set is cocomplete (i. e. that it has small colimits). Note that the correct use of universe levels is not checked by UniMath—we hence have to keep track of them ourselves. This is particularly important when taking (co)limits in Set; we ensure that we only take such (co)limits indexed by a small graph, i. e., a graph whose types of nodes and vertices are elements of U_0.

2 From Multi-Sorted Binding Signatures to Signatures with Strength

In this section we give a category-theoretic notion of abstract signature in the form of strong functors. But first we will consider our main source of examples: signature functors arising from a notion of multi-sorted binding signatures.

2.1 Multi-Sorted Binding Signatures

In what follows, we fix a type S representing the sorts. Many of our constructions will rely on S being a set; in the formalization, this assumption is explicitly stated when needed, but in this paper we take S to be a set everywhere, for sake of simplicity.

We start with the following definition of multi-sorted binding signatures:

Definition 2.1 (MultiSortedSig)**.** A *multi-sorted binding signature* is given by a small set I together with an arity function $\mathsf{ar} : I \to \mathsf{list}(\mathsf{list}(\mathsf{S}) \times \mathsf{S}) \times \mathsf{S}$.

Intuitively, for any $i : I$, there is a term constructor described by its "arity" $\mathsf{ar}(i)$, whose first component describes the list of arguments: for each argument we specify a list of sorts for the variables bound in it, as well as its sort. The second component of $\mathsf{ar}(i)$ designates the sort of the term constructed by the constructor pertaining to the index i. This definition of multi-sorted signatures is not new, and variations on it can be found in the literature, e.g. in the work of Ahrens and Zsidó [14, Definition 4.3]. The standard example is that of the simply-typed λ-calculus (STLC).

Example 2.2 (STLC_Sig)**.** Assume that S is closed under a binary operation $\Rightarrow : \mathsf{S} \to \mathsf{S} \to \mathsf{S}$ representing function types. We have to put into I the sort parameters of the typing rules of the term constructors of STLC. Thus, I is taken to be $(\mathsf{S}\times\mathsf{S})+(\mathsf{S}\times\mathsf{S})$. The left summand pertains to the application operation while the right summand describes λ-abstraction:

$$\mathsf{ar}(\mathsf{inl}\langle s, t\rangle) :\equiv \left\langle [\langle [], s \Rightarrow t\rangle, \langle [], s\rangle], t \right\rangle \ ,$$
$$\mathsf{ar}(\mathsf{inr}\langle s, t\rangle) :\equiv \left\langle [\langle [s], t\rangle], s \Rightarrow t \right\rangle \ .$$

For sorts s and t, there is an application constructor taking as input two terms, of sorts $s \Rightarrow t$ and s, respectively, to yield a term of sort t. Again, for sorts s and t, there is an abstraction constructor taking as input a term of sort t with

a an additional free variable of sort s, and yielding a term of sort $s \Rightarrow t$. This is close to the usual presentation of the typing rules of STLC. However, we do not include a constructor for variables as these will be added generically in Section 2.3. Note that this is a form of locally nameless presentation: bound variables have canonical names.

2.2 Functors from Multi-Sorted Binding Signatures

Recall that C^{S} is the functor category $[\mathsf{S}, C]$ where S is viewed as a discrete category. In the case when C is Set, objects of this categories are simply functions $X : \mathsf{S} \to \mathrm{Set}$. These hence correspond to sets of sorted variables analogously to the parameter X of the example of the inductive family STLC in Section 1. Equivalently, we could consider families of objects in C indexed over S, but $[\mathsf{S}, C]$ has the advantage that results on functor categories (constructions of (co)limits, etc) apply directly.

To construct H, we assume that S is a set and that C has a terminal object 1, binary products, and set-indexed coproducts. We first define a few helper functors:

Definition 2.3 (sorted_option_functor)**.** Let s be a sort. The *sorted option functor* $\mathrm{option}_s : C^{\mathsf{S}} \to C^{\mathsf{S}}$ is defined as

$$\mathrm{option}_s\ X\ t :\equiv X\ t\ +\ \coprod_{(s=t)} 1 \quad .$$

Remark 2.4. The rationale behind the definition is that when given $X : C^{\mathsf{S}}$ and $t : \mathsf{S}$ we get that $\mathrm{option}_s\ X\ t$ is X with an extra variable of sort s if s and t are equal. If S has decidable equality, $=_{\mathrm{dec}}$, this can be defined as

$$\mathrm{option}_s\ X\ t :\equiv \mathrm{if}\ (s =_{\mathrm{dec}} t)\ \mathrm{then}\ (X\ t + 1)\ \mathrm{else}\ (X\ t) \quad .$$

However, in order to avoid assuming decidable equality for S we do the above trick where we form a coproduct of 1 over the type of proofs that $s = t$ (i. e., we form a subsingleton). For this coproduct to exist, e. g., in the category of sets, it would suffice to assume that S is a (small) 1-type so that $s = t$ is a (small) set. However, this would make $\mathrm{option}_s\ X\ t$ add as many variables as there are proofs of $s = t$, which is not what we intend with the definition.

Definition 2.5 (option_list)**.** Given a non-empty list of sorts $\ell \equiv [s_1, \ldots, s_n]$, we define option $\ell : C^{\mathsf{S}} \to C^{\mathsf{S}}$ as

$$\mathrm{option}\ \ell :\equiv \mathrm{option}_{s_1} \cdot (\mathrm{option}_{s_2} \cdot \ldots) \quad .$$

For an empty list, we define option $[] :\equiv \mathrm{Id}$.

Definition 2.6 (projSortToC)**.** For any $s : \mathsf{S}$ we have a *projection functor* $\mathrm{pr}_s : C^{\mathsf{S}} \to C$ defined as:

$$\mathrm{pr}_s\ X :\equiv X\ s \quad .$$

Definition 2.7 (hat_functor)**.** For any $s : \mathsf{S}$ we have a left adjoint[2] to pr_s, written $\hat{s} : C \to C^{\mathsf{S}}$, defined as

$$\hat{s}\ X\ t :\equiv \coprod_{(s=t)} X \quad .$$

[2]The fact that these functors are adjoint is proved in Lemma 3.5.

Remark 2.8. Once again we use the "coproduct-trick" to avoid assuming decidable equality on S. If we had decidable equality then $\hat{s}$ could be defined as

$$\hat{s}\ X\ t :\equiv \mathrm{if}\ (s =_{\mathrm{dec}} t)\ \mathrm{then}\ X\ \mathrm{else}\ 0 \quad ,$$

where 0 is the initial object in C (which exists since C has set-indexed coproducts).

We can now define the functor H in multiple steps. Given $a = (\ell, s) : \mathrm{list}(\mathsf{S}) \times \mathsf{S}$ specifying the sorts of bound variables and type of an argument to some constructor we first define the object part of a functor $F^a : [C^{\mathsf{S}}, C^{\mathsf{S}}] \to [C^{\mathsf{S}}, C]$ as

$$F^a\ X :\equiv \mathrm{pr}_s \cdot X \cdot \mathrm{option}\ \ell \quad .$$

That is, F^a itself is the composition of "precomposition with option ℓ" and "postcomposition with pr_s". Note that F^a is *not* an endofunctor on a category of endofunctors. This will make it necessary, in the next section, to give a general notion of signature with strength to be able to analyze F^a.

Given an arity $(\vec{a}, t)$ with $t : \mathsf{S}$ and $\vec{a} = [a_1, \ldots, a_m]$ with each $a_i = (\ell, s)$ as above, we define the object part of an endofunctor $F^{(\vec{a},t)}$ on $[C^{\mathsf{S}}, C^{\mathsf{S}}]$ as

$$F^{(\vec{a},t)}\ X :\equiv \hat{t} \cdot (F^{a_1}\ X \times \ldots \times F^{a_m}\ X) \quad .$$

More formally, $F^{(\vec{a},t)}$ is obtained by first forming the pointwise product of $F^{a_1}, \ldots, F^{a_m}$ and then by composing with "postcomposition with $\hat{t}$". In the case when $\vec{a}$ is empty, we compose $\hat{t} \cdot \_$ with the functor that constantly outputs 1.

Combining all of this we get the definition of H:

Definition 2.9 (MultiSortedSigToFunctor)**.** Given a multi-sorted binding signature (I, ar), its associated signature functor $H : [C^{\mathsf{S}}, C^{\mathsf{S}}] \to [C^{\mathsf{S}}, C^{\mathsf{S}}]$ is given by the following (pointwise) coproduct:

$$H\ X :\equiv \coprod_{i\ :\ I} F^{\mathrm{ar}(i)}(X) \quad .$$

Note that the above coproduct exists as I is assumed to be a set in Definition 2.1.

Example 2.10 (app_source, lam_source)**.** We apply the general construction and obtain a functor which is the coproduct of two families of functors, app and lam, indexed by $s, t : \mathsf{S}$. These functors are defined pointwise at $X : [C^{\mathsf{S}}, C^{\mathsf{S}}]$ as:

$$\mathrm{app}\ X :\equiv \hat{t} \cdot ((\mathrm{pr}_{s \Rightarrow t} \cdot X) \times (\mathrm{pr}_s \cdot X)) \quad ,$$
$$\mathrm{lam}\ X :\equiv \widehat{s \Rightarrow t} \cdot (\mathrm{pr}_t \cdot X \cdot \mathrm{option}_s) \quad .$$

To obtain exactly these functors in the formalization, special care is needed in $F^{(\vec{a},t)}$ to avoid taking the product with the constant functor. These details make it a little less direct to formalize the strength laws and the proof of ω-cocontinuity; we refer the interested reader to the formalization.

Remark 2.11. There are operations for taking the disjoint sum of signatures, on both multi-sorted binding signatures and signatures with strength. They can be used to assemble complicated signatures from simpler ones, as we do in the

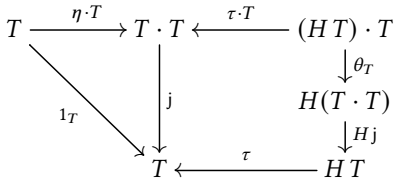

Figure 1. bracket_property_parts_identity_nicer

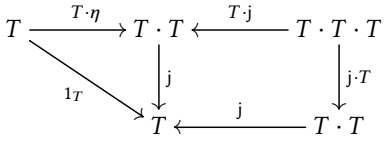

Figure 2. Second and third part of Monad_laws_pointfree

example of PCF in Example 3.9. The translation of signatures presented here preserves these sums.

2.3 Extending the Signature Functor for Substitution

Since variables play a role that is different from all other term constructors when it comes to substitution, the endofunctor H is not supposed to comprise the inclusion of variables into the terms. The "variable case" is added explicitly to H, by considering the endofunctor $\underline{\mathsf{Id}} + H$ on $[C^{\mathrm{S}}, C^{\mathrm{S}}]$. Here $\underline{\mathsf{Id}}$ is the functor that is constantly Id. So on objects, $\underline{\mathsf{Id}} + H$ associates with $X : [C^{\mathrm{S}}, C^{\mathrm{S}}]$ the endofunctor $\mathsf{Id} + H\,X$ on C^{S}. Accordingly, an $(\underline{\mathsf{Id}} + H)$-algebra consists of $T : [C^{\mathrm{S}}, C^{\mathrm{S}}]$ and $\alpha : \mathsf{Id} + H\,T \to T$. It is convenient to present α (uniquely) as $[\eta, \tau]$ with $\eta : \mathsf{Id} \to T$ and $\tau : H\,T \to T$.[3]

We will express substitution in the style of a monad in monoid form, thus with the aforementioned T and η, and moreover a natural transformation $\mathsf{j} : T \cdot T \to T$ that is typically called μ in the literature. We use the triple format since we want to base our implementation on that of [13] which in turn relies on the development of [46] that uses monoid style monads.

In the following definition, we do not care about all the monad laws, but about the specification of the behavior of j (in other words, the "functional properties" that characterize it). Crucially, we address the desideratum that substitution is "homomorphic" on the domain-specific constructors—as specified by the multi-sorted binding signature. The next definition is in the spirit of [46], but isolates more clearly what is needed to specify the properties of substitution itself.

Definition 2.12 (heterogeneous_substitution)**.** Given H as specified above, a **heterogeneous substitution** consists of an $(\underline{\mathsf{Id}} + H)$-algebra (T, η, τ) and a natural transformation

$$\theta : (H-) \cdot T \dashrightarrow H(- \cdot T) : [C^{\mathrm{S}}, C^{\mathrm{S}}] \to [C^{\mathrm{S}}, C^{\mathrm{S}}]$$

between functors of the same type as H, such that there is a unique $[C^{\mathrm{S}}, C^{\mathrm{S}}]$-morphism $\mathsf{j} : T \cdot T \dashrightarrow T$ making the diagram in Fig. 1 commute.

Notice that the functor T in the upper left corner of Fig. 1 is equal to $\mathsf{Id} \cdot T$, in a way which makes the type of arrows out of both convertible, and this allows to type-check $\eta \cdot T$.

We recognize, in the triangle part of the diagram, one of the monad laws—the law saying, intuitively, that substitution for variables is done by look-up. The rectangle part is the specification of "homomorphic" behavior of j on the other term constructors, as instructed by θ. The natural transformation θ is only used with parameter T, but enters fully into the later theorems on unique existence of j when (T, η, τ) is an initial algebra. The notion is not limited to initial algebras ([46] develops the theory for non-wellfounded syntax as well), but we will only use it for those. We do not claim that this data suffices to obtain the other monad laws that are shown in Fig. 2. Note that the functor T in the upper left corner is equal to $T \cdot \mathsf{Id}$, again in a way which allows to type-check $T \cdot \eta$. Likewise, we use associativity of functor composition in the upper right corner that holds in a way that allows to type-check the two arrows out of it, for both possible associations of the expression. If the monad laws are satisfied, the usual name for j is "join".

2.4 Adding Strength to Ensure the Monad Laws

As mentioned above, the unique existence of j according to Definition 2.12 will not suffice to guarantee that (T, η, j) becomes a monad. In this section, we take three simple steps to ensure that our signature functor H does generate a monad. The second and third step are intertwined, and are discussed together.

2.4.1 Generalizing the Type of θ. Firstly, more natural transformations have to be uniquely determined by a generalization of Fig. 1 which depends on a version of θ with two arguments. Given a signature functor H as before, we will call *prestrength* for H any natural transformation $\theta : (H-) \cdot U\sim \to H(- \cdot U\sim)$ between bifunctors $[C^{\mathrm{S}}, C^{\mathrm{S}}] \times \mathrm{Ptd}(C^{\mathrm{S}}) \to [C^{\mathrm{S}}, C^{\mathrm{S}}]$, with the category $\mathrm{Ptd}(C^{\mathrm{S}})$ of pointed endofunctors over C^{S} and its forgetful functor $U : \mathrm{Ptd}(C^{\mathrm{S}}) \to [C^{\mathrm{S}}, C^{\mathrm{S}}]$. This notion of prestrength will be an instance of the less constrained Definition 2.14 below. In the situation of Definition 2.12, we have the pointed endofunctor (T, η), and if we fix the second argument (symbolized by $\sim$) of a prestrength θ to (T, η), we get (by virtue of $U(T, \eta) \equiv T$) a natural transformation of the type required in that definition.

Definition 2.13 (hss)**.** Given H as before and a prestrength θ for H, an $(\underline{\mathsf{Id}} + H)$-algebra (T, η, τ) is a **heterogeneous substitution system** (HSS) for (H, θ), if, for every $\mathrm{Ptd}(C^{\mathrm{S}})$-morphism $f : (Z, e) \to (T, \eta)$, there exists a unique $[C^{\mathrm{S}}, C^{\mathrm{S}}]$-morphism $h : T \cdot Z \to T$ making the diagram in Fig. 3 commute. This morphism is denoted $(\!|f|\!)$.

[3] Note that we are not assuming that (T, α) is an *initial* $(\underline{\mathsf{Id}} + H)$-algebra. The construction of initial $(\underline{\mathsf{Id}} + H)$-algebras is considered in Section 3.

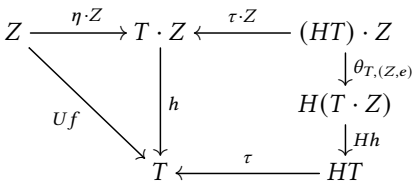

Figure 3. bracket_property_parts

Given our previous remark on fixing the second argument of θ to (T, η), the uniquely existing j of a heterogeneous substitution is the special case with $f :\equiv \mathsf{id}_{(T,\eta)}$ (i. e., the identity natural transformation). Analogously to Definition 2.12, the first argument of θ is always set to T in this diagram, but needs to vary when proving that one obtains an HSS when (T, η, τ) is an initial algebra.

2.4.2 Widening the Type of H and Adding Strength Laws to θ. Secondly, after this first step, we will still not be able to ensure the monad laws for the obtained (T, η, j) through an HSS. We are lacking laws for θ concerning its dependency on its second argument. However, there is a third concern of a more practical nature: if we take the first step, we have to define a prestrength θ for the signature functor H generated in Section 2.2 along the structure of the given multi-sorted binding signature. As already mentioned there, the functor $F^a : [C^{\mathrm{S}}, C^{\mathrm{S}}] \to [C^{\mathrm{S}}, C]$, which is a building block of that construction of H, does not have the same type as H, and it is not even an endofunctor. So, if we are to develop a library of functors with prestrength, we need to widen the notion so as to cover those situation with "varying types" as well. We will then be able to construct prestrengths for all building blocks of the signature functor H. In parallel, we can propagate suitable laws for them so that the laws governing the prestrength for H ensure that an HSS for it will satisfy the monad laws (see Lemma 2.16 below).

Definition 2.14 (PrestrengthForSignature)**.** For categories C, $\mathcal{D}$, $\mathcal{D}'$ and a functor $H : [C, \mathcal{D}'] \to [C, \mathcal{D}]$, a **prestrength** for H is a natural transformation

$$\theta : (H-) \cdot U\sim \to H(- \cdot U\sim)$$

between bifunctors $[C, \mathcal{D}'] \times \mathrm{Ptd}(C) \to [C, \mathcal{D}]$.

Definition 2.15 (StrengthForSignature)**.** Let a prestrength θ be given for a functor H as specified above. It is called a **strength** for H if it is "homomorphic" in the second argument $\sim$, in the following sense. The source and target bifunctors applied to a pair of objects $(X, (Z, e))$ with $X : [C, \mathcal{D}']$ and $(Z, e) : \mathrm{Ptd}(C)$ (X for the argument symbolized by $-$ and (Z, e) for the argument symbolized by $\sim$) yield $HX \cdot Z$ and $H(X \cdot Z)$, thus $\theta_{X,(Z,e)} : HX \cdot Z \to H(X \cdot Z)$ in $[C, \mathcal{D}]$. Being "homomorphic" of θ in the second argument for us means satisfying the equations[4] $\theta_{X,\mathsf{id}} = \mathsf{id}_{HX}$ and

$$\theta_{X,(Z' \cdot Z, e' \cdot e)} = H(\alpha^{-1}_{X,Z',Z}) \circ \theta_{X \cdot Z',(Z,e)} \circ (\theta_{X,(Z',e')} \cdot Z) \ ,$$

[4]See theta_Strength1_int_nicer and theta_Strength2_int_nicest.

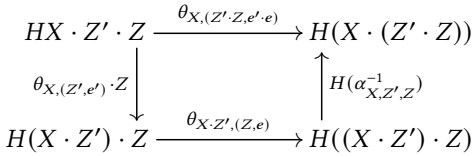

Figure 4. Second strength law for signatures

where $\alpha_{X,Y,Z} : X \cdot (Y \cdot Z) \simeq (X \cdot Y) \cdot Z$ is the associator. The equation is illustrated by the diagram in Fig. 4. When θ is a strength for H, the pair (H, θ) is called a **signature with strength**.

Analogously to Fig. 2, the upper left corner can be associated either way for the type-checking of arrows out of it. On the right-hand side, we cannot escape using the "monoidal isomorphism" α witnessing associativity since the application of H breaks the convertibility of the types of arrows into $H(X \cdot (Z' \cdot Z))$ and $H((X \cdot Z') \cdot Z)$. Since X is not assumed to be an endomorphism, we use the notion of "monoidal isomorphism" by analogy only. Mathematically, the arrow to the right is just the identity. Approximatively, the diagram expresses that θ with a composition in the second argument is the composition of instances of θ for each of the composites in the second argument. This looks like a homomorphism of monoids, with $\mathrm{Ptd}(C)$ equipped with the identity and the associative operation of composition, and the first equation yields an identity, but the second equation varies also the first argument to θ. In Section 4 we shed more light on this notion and in which sense this is "homomorphic".

In the case where $\mathcal{D}$ and $\mathcal{D}'$ are just C, all functor compositions take place in $[C, C]$, and we precisely get back the strength concept of [46], with the formalization-dictated addition of the monoidal isomorphism that is part of the formulation of [12]. In the formalization, we follow [12] and have all monoidal isomorphisms explicit. A contribution of the present paper is the identification of the "widening" of the concept to three possibly different parameter categories C, $\mathcal{D}$ and $\mathcal{D}'$, as dictated by the necessity of a modular definition of the strength for the signature functor generated by a *multi-sorted* binding signature. This formal widening is a conceptual step, but its implementation was direct on the basis of the formalization provided by [12]. Basically, all the proofs worked without changes other than the adaptation to the widening of the statement. Such a "refactoring" step is thus painless given a full formalization.

It does not seem meaningful for us to form fixed points with signatures with strength, unless the three parameter categories are identical: the prominent role of pointed endofunctors on C in the laws makes them unapplicable for the analysis of other functors that might arise as fixed points. So, otherwise, we consider those H and their strength only as building blocks.

As the notion of strength for H appearing in the definition of HSS is unaltered, we can simply use [46, Theorem 10], formalized as [12, Theorem 26] that now reads as:

Lemma 2.16 (Monad_from_hss). *Let C be a category with binary coproducts, and let H be an endofunctor on $[C, C]$ and let θ be such that (H, θ) is a signature with strength. If an $(\underline{\mathrm{Id}} + H)$-algebra $(T, [\eta, \tau])$ is an HSS, then the definition*

$$\mathsf{join} :\equiv (\!|1_{(T,\eta)}|\!) : T \cdot T \to T$$

yields a monad (T, η, join) in monoid form.

In order to construct the strength from the building blocks of the generated signature functor H in Section 2.2, we will also use the concept of pointed distributive law from [13, Definition 10]. We avoid monoidal isomorphisms and give the following definition.

Definition 2.17 (DistributiveLaw). Given $G : [C, C]$ for a category C, a **pointed distributive law** is a natural transformation $\delta : G \cdot U{\sim} \to U{\sim} \cdot G$ of functors $\mathrm{Ptd}(C) \to [C, C]$ such that $\delta_{\mathrm{Id}} = \mathrm{id}_G$ and $\delta_{(Z' \cdot Z, e' \cdot e)} = Z' \cdot \delta_{(Z,e)} \circ \delta_{(Z',e')} \cdot Z$.[5]

We now sketch the construction of the strength for the obtained signature functor. For option_s, construct a pointed distributive law, compose those distributive laws to obtain one for option ℓ and then generate the strength for precomposition with option ℓ. Pointwise post-composition with a fixed functor (here with pr_t) generically allows us to construct a strength from the given one, which is good for F^a. This is a strength in the wide sense introduced in this paper.

The product of finitely many such F^{a_i} can be treated easily by using the corresponding construction of strengths for binary products iteratively. Let us mention the following implementation detail: the constructed signature with strength ought to have, as first component, the product of those F^{a_i} w. r. t. *convertibility* and not just provably. For one construction step, there is no concern, but this has to hold even for the iteration process, so the definitions have to go through the structure twice (as seen in Sig_exp_functor_list).

The strength construction for $F^{(\vec{a},t)}$ is then dealt with by another instance of the strength construction for pointwise postcomposition with a fixed functor, this time $\hat{t}$. Canonically, the strength carries over to coproducts and thus to the H associated to a multi-sorted binding signature.

Following the steps described above, we construct a signature with strength from the given multi-sorted binding signature. We have thus a full specification: starting from a multi-sorted binding signature, we construct a signature with strength. The only remaining task is to construct an $(\underline{\mathrm{Id}} + H)$-algebra representing terms that is, in particular, an HSS. This then provides us with j as monad multiplication and thus a certified monadic substitution operation.

[5]The second equation is delta_law2_nicer.

3 From Multi-Sorted Binding Signatures to Certified Substitution

Having constructed the signature with strength (H, θ) from a multi-sorted binding signature, we are now ready to construct the monadic substitution operation. The key theorem for this is:

Theorem 3.1 (TermMonad). *Let (H, θ) be a signature with strength, where $H : [C, C] \to [C, C]$ is ω-cocontinuous and C has binary coproducts, an initial object, and colimits of chains. Then the initial $(\underline{\mathrm{Id}} + H)$-algebra exists and is equipped with a monad structure.*

The constructions in the proof of this theorem were given by [13] and go via the heterogeneous substitution systems of Lemma 2.16. For the purposes of the present paper we treat Theorem 3.1 as a black box, but stating it this way makes it applicable to any abstract signature (H, θ) satisfying the assumptions. This way the result is decoupled from a specific notion of concrete signature and applies both to the binding signatures of [13] and those in Section 2.1. It should also apply to other languages which are not instances of these notions of signatures. An example is the λ-calculus with explicit flattening as in Matthes and Uustalu [46], but ω-cocontinuity of this has not yet been addressed properly and is left as an open question. We now turn our attention to proving that H, as defined in Section 2.2, is ω-cocontinuous.

3.1 Proving ω-Cocontinuity of the Signature Functor

We start off by recalling the definition of ω-cocontinuity in UniMath. Colimits are parametrized by diagrams over graphs, as suggested by Mac Lane [43, p. 71]. A functor $F : C \to \mathcal{D}$ *preserves* colimits of shape G if, for any diagram d of shape G in C, and any cocone a under d with tip C, the cocone $F\,a$ is colimiting for $F\,d$ whenever a is colimiting for d.

A functor is called *ω-cocontinuous* (is_omega_cocont) if it preserves colimits of diagrams of the shape

$$A_0 \xrightarrow{f_0} A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} \dots$$

that is, diagrams on the graph where objects are natural numbers and where there is a unique arrow from m to n if and only if $1 + m = n$. We refer to diagrams of this shape as *chains*. Actually, in UniMath, the type of arrows from m to n is *defined* to be the type of proofs that $1 + m = n$, exploiting that the type of natural numbers is a set.

In order to prove that the signature functor H of Section 2.2 satisfies is_omega_cocont we will apply lemmas that decompose the goal into smaller pieces. In particular, the proof will boil down to proving that "postcomposition with pr_s" and "postcomposition with $\hat{t}$" are ω-cocontinuous. Along the way we identify sufficient conditions on C.

The general form of the following lemmas were added to UniMath by [13]. For simplicity we only recall them in the form that we need them in this paper:

Lemma 3.2 (Examples of ω-cocontinuous functors)**.**

1. *Any constant functor $\underline{d} : \mathcal{C} \to \mathcal{D}$ is ω-cocontinuous (is_omega_cocont_constant_functor).*
2. *The composition of ω-cocontinuous functors is again ω-cocontinuous (is_omega_cocont_functor_composite).*
3. *Let $\mathcal{C}$ and $\mathcal{D}$ be categories with specified binary products and further assume that $d \times -$ is ω-cocontinuous for all $d : \mathcal{D}_0$. The binary product, $F \times G : \mathcal{C} \to \mathcal{D}$, of ω-cocontinuous functors $F, G : \mathcal{C} \to \mathcal{D}$ is ω-cocontinuous (is_omega_cocont_BinProduct_of_functors).*
4. *Let $\mathcal{C}$ be a category and $\mathcal{D}$ a category with specified coproducts. Given an I-indexed family of functors $F_i : \mathcal{C} \to \mathcal{D}$ the coproduct $\coprod_{i:I} F_i : \mathcal{C} \to \mathcal{D}$ is ω-cocontinuous (is_omega_cocont_coproduct_of_functors).*
5. *Let $\mathcal{C}$ have colimits of chains and let $F : \mathcal{A} \to \mathcal{B}$ be a functor, then the functor "precomposition with F", $\_ \cdot F : [\mathcal{B}, \mathcal{C}] \to [\mathcal{A}, \mathcal{C}]$, is ω-cocontinuous (is_omega_cocont_pre_composition_functor).*

By using point 4 we can decompose ω-cocontinuity of H to showing that each $F^{(\vec{a},t)}$ is ω-cocontinuous. By point 2 and repeated application of 3 we now need to show that $\hat{t} \cdot \_$ and F^{a_i} are ω-cocontinuous. Point 1 is used to handle the case when $\vec{a}$ is empty. By points 2 and 5 the fact that each F^{a_i} is ω-cocontinuous is reduced to ω-cocontinuity of $\mathrm{pr}_s \cdot \_$.

As limits and colimits are computed pointwise in the functor category $[\mathcal{C}^{\mathrm{S}}, \mathcal{C}^{\mathrm{S}}]$ we see that the assumptions on $\mathcal{C}$ for making the above argument go through is that $\mathcal{C}$ has specified binary products, coproducts, colimits of chains, and that $F \times -$ is ω-cocontinuous for all $F : [\mathcal{C}^{\mathrm{S}}, \mathcal{C}]$.

The important facts that are not yet covered by the above argument are the ω-cocontinuity of $\mathrm{pr}_t \cdot \_$ and $\hat{t} \cdot \_$. The key for proving these is:

Lemma 3.3 (left_adjoint_cocont)**.** *If $F : \mathcal{C} \to \mathcal{D}$ is a left adjoint then it preserves colimits.*

This classic result in category theory could also be found in `UniMath` and it implies that any left adjoint functor is ω-cocontinuous. However, `UniMath` did not provide any lemmas about the ω-cocontinuity of postcomposition with various functors. We hence formalized the following lemma:

Lemma 3.4 (is_left_adjoint_post_composition_functor)**.** *If $F : \mathcal{A} \to \mathcal{B}$ is a left adjoint then $F \cdot \_ : [\mathcal{C}, \mathcal{A}] \to [\mathcal{C}, \mathcal{B}]$ is a left adjoint.*

Hence, if F is a left adjoint, then $F \cdot \_$ is ω-cocontinuous (is_omega_cocont_post_composition_functor). As remarked without proof in Section 2.2, the functor $\hat{t}$ is left adjoint to pr_t, so $\hat{t} \cdot \_$ is ω-cocontinuous. We will now establish this adjunction as well as the fact that pr_t also has a right adjoint, which implies that $\mathrm{pr}_t \cdot \_$ is ω-cocontinuous.

In order to provide some intuition for the below proof, note that pr_t has a more abstract description. If we write t also for the map $1 \to \mathrm{S}$ picking out the sort t, then "precomposition with t" is a functor $\_ \cdot t : \mathcal{C}^{\mathrm{S}} \to \mathcal{C}^{1}$. It is easy to see that this is isomorphic to pr_t, so if we assume that $\mathcal{C}$ has suitable (co)limits then it will have both left and right adjoints given by Kan extension. This abstract argument indicates that we should indeed be able to establish that pr_t has both left and right adjoints given that $\mathcal{C}$ has suitable (co)limits. However, we can characterize these much more concretely as follows:

Lemma 3.5 (is_left_adjoint_hat)**.** *Let $\mathcal{C}$ be a category with specified set-indexed coproducts. The functor pr_t has a left adjoint $\hat{t} : \mathcal{C} \to \mathcal{C}^{\mathrm{S}}$ defined as*

$$\hat{t}\, X\, s :\equiv \coprod_{(t=s)} X \ .$$

Lemma 3.6 (is_left_adjoint_projSortToC)**.** *Let $\mathcal{C}$ be a category with specified products. The functor pr_t has a right adjoint $\underline{t} : \mathcal{C} \to \mathcal{C}^{\mathrm{S}}$ defined as*

$$\underline{t}\, A\, s :\equiv \prod_{(t=s)} A \ .$$

The reason we require the coproducts to be set-indexed is that this is needed for the existence of H (recall that I is assumed to be a set). For Lemma 3.6 it would have sufficed to assume existence of proposition-indexed products, however, the stronger assumption of arbitrary (small) products is usually not a problem as the existence of such products in concrete categories typically does not depend on the homotopy level of the indexing type. For instance, Set has products indexed by arbitrary (small) types, but only set-indexed coproducts. Combining all of this we get:

Theorem 3.7 (omega_cocont_MultiSortedSigToSig)**.** *Given a category $\mathcal{C}$ with chosen products, set-indexed coproducts, colimits of chains, such that $F \times -$ is ω-cocontinuous for all $F : [\mathcal{C}^{\mathrm{S}}, \mathcal{C}]$, then the signature functor H associated to a binding signature is ω-cocontinuous.*

In the formalization we provide a slightly different interface to the construction than the one of Theorem 3.7. In particular, we assume that $\mathcal{C}$ has chosen terminal and initial objects as well as binary (co)products. The reason for this is that, even though these assumption follow from the assumptions of the theorem, one can often give more direct constructions of these assumptions in concrete categories.

3.2 Instantiating the Framework and Examples

We can now instantiate Theorem 3.7 and then Theorem 3.1 with $\mathcal{C} = \mathrm{Set}$. The `UniMath` library contains proofs that Set is (co)complete as well as lemmas about (co)limits in functor categories, so most assumptions are easily satisfied. Proving that Set has colimits is in fact one of the places where we profit from working in univalent type theory. This construction relies on set quotients which are not directly available in intensional type theory, for details see [13, Section 3.3]. The last assumption of Theorem 3.7 is satisfied if $[\mathcal{C}^{\mathrm{S}}, \mathcal{C}]$ has exponentials. These are not as easy to compute in functor

categories as (co)limits, luckily UniMath already has a formalization of them when the target category is Set. We can hence instantiate the framework and obtain some examples.

Example 3.8 (SubstitutionSystems.STLC_alt.v)**.** Assuming that S is closed under a binary operation $\Rightarrow\colon \mathsf{S} \to \mathsf{S} \to \mathsf{S}$, we have the multi-sorted signature of STLC from Example 2.2. Applying the construction of Section 2.2 to get H as in Example 2.10, and providing it, together with its strength θ, to Theorem 3.1, we obtain an initial algebra $\Lambda : [\mathrm{Set}^{\mathsf{S}}, \mathrm{Set}^{\mathsf{S}}]$ for H together with a monadic substitution operation. Given $X, Y : \mathrm{Set}^{\mathsf{S}}$ and a morphism $f : X \to \Lambda\, Y$, the substitution operation yields a morphism $\mathrm{subst}\, f : \Lambda\,(X+Y) \to \Lambda\, Y$. This is a parallel substitution, replacing all occurrences of X in one go. By instantiating with 1 for X we obtain an operation which just replaces the occurrence of one variable.

The monad laws state that subst is well-behaved and from them we can derive various standard laws for substitution. For example, the interchange law for commuting two substitutions can be derived generically for any monad M. By instantiation we then obtain that subst satisfies this law. For details, see the formal development.

We have also implemented more complex languages, including Plotkin's PCF [50] and the pre-syntax of the Calculus of Constructions (CoC) à la Streicher [57]. For space restrictions we refer to the formalization for details.

Example 3.9 (SubstitutionSystems.PCF_alt.v)**.** PCF is an extension of STLC with natural number and boolean types, together with operations on these. Specifying it as a multi-sorted binding signature is direct and uses the disjoint sum of signatures to extend the signature of STLC.

Example 3.10 (SubstitutionSystems.CCS_alt.v)**.** In [57] the CoC is presented as a 2-sorted language with $\mathsf{S} = \{\mathsf{ty}, \mathsf{el}\}$ representing the types and terms. The pre-syntax part of this is a simple multi-sorted binding signature, with binders for both types and terms. Formalizing it posed no difficulties.

4 Understanding the Notion of Strength

Originally, the aim of the work presented in this paper was to benefit from our previous construction of a certified monadic substitution for *untyped* terms based on ω-cocontinuous base functors H with strength, which is also implemented in UniMath [13]. The move is to a more complex base category to account for *multi-sorted* terms, the construction of H now from a *multi-sorted* binding signature. But for the construction of strength of H to be modular, we identified the need for the widening of the strength concept to account for functors that are not endo. This all worked, in the sense of providing computer-formalized mathematical structures, with a certified monadic substitution operation as outcome. The original strength concept had already been understood mathematically, in the sense of relating it to more abstract mathematical notions, by Ahrens and Matthes [12]: they propose a definition of *relative strength* [12, Definition 11] that is spelt out on the level of monoidal categories, but directed towards the implemented strength. However, the authors sketched the instantiation very briefly and also mentioned that this is an instance of a strength concept for actions. The definition of relative strength just mentioned ("relative" does not suggest that we are aiming for *relative monads*), and also the use of action strength by Fiore [26], were geared towards uses with a specific monoidal category, the one canonically obtained through composition of endomorphisms.

In what follows, we use two other dimensions of UniMath beyond the library of certified structures: we are formalizing the meta-theory behind the concepts, in particular, we prove the claims in [12] as to the link with abstract strength concepts. We also use it as a research tool to obtain new results on the mathematical justification of the widened strength notion, here by (i) identifying a bicategorical scenario for action strength, and by (ii) continuing the investigation into a "higher-order" view of actions as monoidal functors, as put forward by Janelidze and Kelly [40]. We even mention work in progress in that spirit that sees action strength itself as a monoidal functor.

4.1 Action-Based Strength

Action-based strength will be our term for the extra requirement on a functor $F : \mathcal{A} \to \mathcal{A}'$ to be a morphism between actions of a monoidal category $\mathcal{V}$ on $\mathcal{A}$ and $\mathcal{A}'$, respectively. We first present the "point-free version" (param_distr_pentagon_eq_body) based on the definition of actions found in [40]. This version has the advantage of asking for very little to be added to "general category theory". However, the "natural" level of generality of its definition is in an arbitrary bicategory instead of the different functor categories, still with an ordinary monoidal category as parameter space, and most of our formal proof development is on that level.

We consider a monoidal category $\mathcal{V}$ with tensor product $\otimes$, categories $\mathcal{A}$ and $\mathcal{A}'$ and actions of $\mathcal{V}$ on both, expressed as strong monoidal functors $F : \mathcal{V} \to [\mathcal{A}, \mathcal{A}]$ and $F' : \mathcal{V} \to [\mathcal{A}', \mathcal{A}']$ (where the monoidal structure on the endofunctor categories is given by composition in diagrammatic order). Let $G : \mathcal{A} \to \mathcal{A}'$ be a functor and let δ be a natural transformation between functors from $\mathcal{V}$ to $[\mathcal{A}, \mathcal{A}']$ that are defined on objects v of $\mathcal{V}$ as $F'v \cdot G$ and $G \cdot Fv$, respectively. The transformation δ is a **parameterized distributivity** for G (but could also be called a *strength* of G) if the diagrams in Fig. 5 commute.

In the diagrams, all nodes are functors from $\mathcal{A}$ to $\mathcal{A}'$, with I the unit of $\mathcal{V}$. The arrows $\epsilon : \mathrm{Id} \to FI$ and $\mu_{v,w} : Fw \cdot Fv \to F(v \otimes w)$ are the isomorphisms coming from F being strong monoidal, and likewise for ϵ' and μ' relative to F'. Since the vertically arranged morphisms in the pentagon diagram are isomorphisms, the diagram is essentially a triangle describing $\delta_{v\otimes w}$ as a suitable composition of δ_v and δ_w,

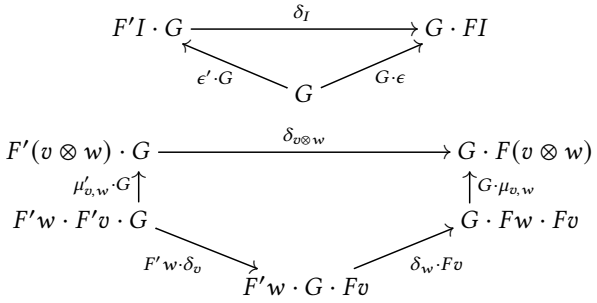

Figure 5. param_distr_triangle_eq, param_distr_pentagon_eq_body

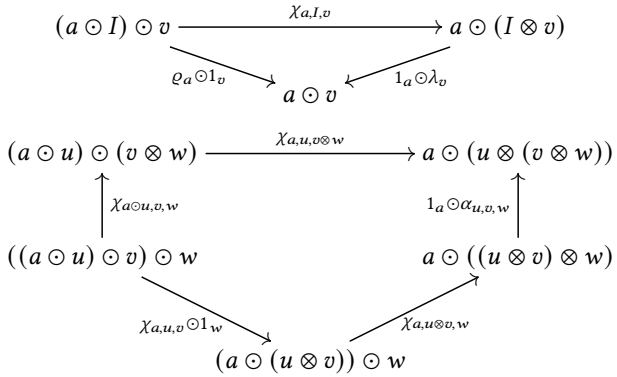

Figure 6. action_triangle_eq, action_pentagon_eq

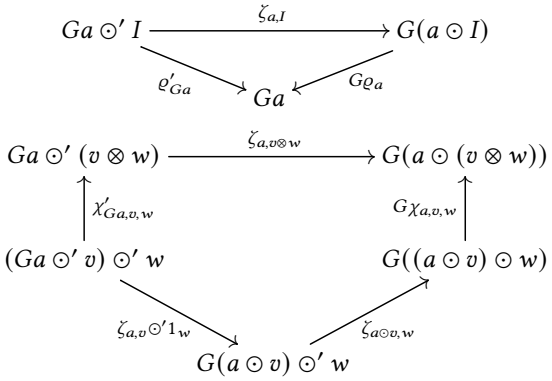

Figure 7. actionbased_strength_triangle_eq, actionbased_strength_pentagon_eq (correspond to those of Fig. 5)

as expected (and in accordance with our notion of strength in Definition 2.15). Let us remark that (G, δ) cannot be seen as a monoidal natural transformation from F to F'—already because they need not have the same target category—so this definition is specific to the interpretation of the strong monoidal functors F and F' as actions.

While the point-free definition is concise, we rather base our analysis of strength for signatures on a more concrete version that is very close to the original definition of Pareigis [49], coming under the name of $\mathcal{C}$-categories for actions and $\mathcal{C}$-functors for strong functors between actions. We even formalized a transformation of action strengths from the point-free to the concrete setting (actionbased_strong_functor_from_alt). The concrete version exploits the equivalence between the functor categories $[\mathcal{A} \times \mathcal{B}, \mathcal{C}]$ and $[\mathcal{B}, [\mathcal{A}, \mathcal{C}]]$ which is currying / uncurrying (together with swapping $\mathcal{A}$ and $\mathcal{B}$). This equivalence extends to a biadjunction in the bicategory of (small) categories (currying_biajd).

Definition 4.1 (action)**.** An **action** of the given monoidal category $\mathcal{V}$ (as above) is a functor $\odot : \mathcal{A} \times \mathcal{C} \to \mathcal{A}$ (instead of its curried version $F : \mathcal{C} \to [\mathcal{A}, \mathcal{A}]$) that is written infix, together with natural isomorphisms ϱ, with components $\varrho_a : a \odot I \to a$ (*right unitor* of the action), and χ, with components $\chi_{a,v,w} : (a \odot v) \odot w \to a \odot (v \otimes w)$ (*convertor* of the action), that make the diagrams in Fig. 6 commute (with λ and α the left unitor and associator of $\mathcal{V}$, respectively).

Definition 4.2 (actionbased_strength)**.** Let $\mathcal{V}$ be a monoidal category, and let $\odot$ and $\odot'$ be V-actions on categories $\mathcal{A}$, $\mathcal{A}'$, with ϱ, ϱ', χ, χ' the right unitors and convertors of the actions, respectively. Let $G : \mathcal{A} \to \mathcal{A}'$ be a functor. A natural transformation ζ, with components $\zeta_{a,v} : Ga \odot' v \to G(a \odot v)$ is an **action-based strength** for G if the diagrams in Fig. 7 commute. The pair (G, ζ) is then called a **strong action-based functor** from $\odot$ to $\odot'$.

Notice that [49] required ζ to be a natural isomorphism, which would make our application to multi-sorted binding signatures impossible (also Fiore [26] worked without the requirement of isomorphism). Already in [49], there is a notion of morphism between strong action-based functors, under the name $\mathcal{C}$-morphism. With our notations:

Definition 4.3 (Natural transformation of strong action-based functors)**.** Given the data in the preceding definition and a further functor $G' : \mathcal{A} \to \mathcal{A}'$ with action-based strength ζ', a natural transformation $\eta : G \to G'$ is an **action-based natural transformation** if the diagram

$$\begin{array}{ccc} Ga \odot' v & \xrightarrow{\zeta_{a,v}} & G(a \odot v) \\ {\scriptstyle \eta_a \odot' 1_v}\downarrow & & \downarrow{\scriptstyle \eta_{a \odot v}} \\ G'a \odot' v & \xrightarrow{\zeta'_{a,v}} & G'(a \odot v) \end{array} \tag{1}$$

commutes (Strong_Functor_Category_mor_diagram).

For a fixed monoidal category $\mathcal{V}$, categories $\mathcal{A}$ and $\mathcal{A}'$ and actions $\odot$ and $\odot'$ of $\mathcal{V}$ on $\mathcal{A}$ and $\mathcal{A}'$, we define the "action-based strong functor category" with the strong action-based functors from $\odot$ to $\odot'$ as objects and the action-based natural transformations between those objects as morphisms. This is comfortably done in `UniMath` by exploiting the concept of displayed category introduced by Ahrens and Lumsdaine [11], meaning that only the extra data and constraints with respect to the underlying functor category $[\mathcal{A}, \mathcal{A}']$ have to be given. This leads to a displayed category (Strong_

Functor_category_displayed), from which the sought category is obtained generically as the "total" category (Strong_Functor_category). In this case, a formalized form of the "structure identity principle" [60, Section 9.8] allows us to prove in a straightforward and modular way that the category thus obtained is univalent when the target category $\mathcal{A}'$ is (is_univalent_Strong_Functor_category). For this to be applicable, it is crucial that the notion of *action-based* natural transformation does not come with supplementary data, just with a property (i. e., a proposition) to be satisfied. This makes it especially rewarding to use the framework of displayed bicategories by Ahrens et al. [6] (as implemented in Bicategories.DisplayedBicats.DispBicat.v) to construct the bicategory of actions, strong action-based functors and action-based natural transformations (for a given monoidal category $\mathcal{V}$) as displayed over the bicategory of (small) categories (actions_disp_bicat) (from which the library immediately derives an "ordinary" bicategory (actions_bicat)). Also notice that the diagram in Eq. (1) is symmetric in the sense that if η is an isomorphism, its inverse satisfies it again (with (G, ζ) and (G', ζ') exchanged). In other words, the obtained displayed bicategory is locally a groupoid (actions_disp_locally_groupoid). We are not aware of such an observation in the literature.

4.2 Signature Strength as Action-Based Strength

In the interest of securing the meta-theory as well, we formalize relative strength of [12, Definition 11] (rel_strength) and formally confirm the claims of [12]: (i) The parameters of action-based strength can be instantiated so that one can construct transformations from relative strength to instantiated action-based strength (from_relative_strength) and vice versa (from_actionbased_strength). (ii) For the case of $\mathcal{D} = \mathcal{D}' = C$ in Definition 2.15, the parameters of relative strength can be instantiated so that one can construct transformations from strength to instantiated relative strength (from_signature) and vice versa (signature_from).

The endofunctors, and the pointed endofunctors, over a category C readily form monoidal categories, and the forgetful functor U that forgets the "points" is even a strong monoidal functor between these monoidal categories (from_ptd_as_strong_monoidal_functor). It is this U that is used in step (ii) above to instantiate the parameter of that name in relative strength (mentioned already in [12]). However, in the notion of prestrength, we also have to deal with functors that are not endos. The notion of action is wide enough to handle this: the endomorphisms of C act on any functor category with source category C, see below. To see the situation of Definition 2.14 more clearly, we abstract away from the bicategory of (small) categories with their functors and natural transformations and assume a bicategory B. The categories C, $\mathcal{D}$ and $\mathcal{D}'$ of Definition 2.14 are replaced by objects c_0, d_0, d_0' of B. The object c_0 gives rise to a monoidal category $\mathrm{Endo}_B(c_0)$ built from the hom-category $B(c_0, c_0)$, i. e., the endomorphisms of c_0 and their 2-cells. $\mathrm{Endo}_B(c_0)$ replaces $[C, C]$ in the inner workings of Definition 2.14, while $\mathrm{Ptd}(C)$ is replaced by an arbitrary monoidal category $\mathcal{V}$, and with an arbitrary strong monoidal U from $\mathcal{V}$ to $\mathrm{Endo}_B(c_0)$.

Now, for any objects c_0, d_0 of B, we can define an action $\odot_{c_0,d_0}$ of $\mathrm{Endo}_B(c_0)$ on the hom-category $B(c_0, d_0)$, which on objects $f : c_0 \to d_0$, $g : c_0 \to c_0$ is given by $f \odot_{c_0,d_0} g :\equiv f \cdot g$. We say that $\mathrm{Endo}_B(c_0)$ *acts by precomposition* on $B(c_0, d_0)$ (together with the right unitor and convertor satisfying the laws in Fig. 6), formalized as action_from_precomp.

Given a strong monoidal functor from $\mathcal{V}$ to $\mathcal{V}'$, any action of $\mathcal{V}'$ can be lifted to an action of $\mathcal{V}$ on the same category (this generalizes an observation by Fiore [26, p. 59] that the (obvious) action of $\mathcal{V}'$ on itself can be lifted that way), formalized as lifted_action. Thus, we get an action of $\mathcal{V}$ on $B(c_0, d_0)$. Reusing this action construction for d_0' in place of d_0 as well, one can study action-based strength for a given functor $G : B(c_0, d_0') \to B(c_0, d_0)$ that replaces H of Definition 2.14. The laws of Fig. 7 can then be instantiated to the present abstract bicategorical scenario (see the logically equivalent formalized triangle_eq_nice and pentagon_eq_nice).

Of course, we want to instantiate all of this to the bicategory CAT of (small) categories, formalized as bicat_of_cats. A small technical problem arises as the forgetful functor U from $\mathrm{Ptd}(C)$ is not seen by the system as having target $\mathrm{Endo}_{\mathrm{CAT}}(C)$. Once this is solved, we identify the equivalence of this instance of action-based strength with the notion of strength in Definition 2.15. This equivalence is not only logical, but extends to an adjoint equivalence of a suitably formed category of signatures with strength (for given C, $\mathcal{D}$, $\mathcal{D}'$) and the instance of the action-based strong functor category mentioned after Definition 4.3, formalized as EquivalenceSignaturesABStrongFunctors.

4.3 Understanding Strength Itself

In category theory, many concepts can be mutually reduced to each other, by appropriately chosen instances for the parameter categories of these concepts. Mac Lane [43] has many examples already for the most common concepts such as initial objects, limits and adjunctions. It is a matter of taste if an action is better "understood" in form of the concrete Definition 4.1 or as a monoidal functor into endomorphisms, but the second concept had already been there. For action strength, one might ask if the extra concept of parameterized distributivity has some merit over the likewise concrete, but prior Definition 4.2 (even if the concepts relate through categorical currying, we consider them as distinct).

The vertical arrows in Fig. 5 are isomorphisms, as are the legs of the triangle, so one might be tempted to say that the laws express that δ is a homomorphism, with δ_I being the "identity", and $\delta_{v \otimes w}$ being the "composition" of δ_v and δ_w. However, the nodes in the pentagon diagram depend on the objects v, w of $\mathcal{V}$, and δ is a natural transformation between functors into a functor category. Since the parameters run

through the monoidal category $\mathcal{V}$, the appropriate notion of homomorphism for δ is necessarily by way of a monoidal functor with source $\mathcal{V}$ to which δ closely corresponds.

We exploit that natural transformations can sometimes be seen as functors and extend this to the representation of parameterized distributivity as strong monoidal functors.

First, we study the elementary situation in plain categories, with a precise connection between, on the one hand, natural transformations of functors into a functor category and, on the other hand, functors into a specifically crafted category. More precisely, natural transformations between $H, H' : C \to [\mathcal{A}, \mathcal{A}']$ correspond to functors from C to a category $\mathcal{T}(H, H')$ (trafotarget_cat) that is the total category of a displayed category over C (trafotarget_disp): an object "over" c is the pair consisting of c and a natural transformation $\alpha^c : H\,c \to H'\,c$—hence a morphism of $[\mathcal{A}, \mathcal{A}']$—and a morphism "over" $f : c \to c'$ is a pair consisting of f and a proof that the "naturality diagram" commutes:

$$\begin{array}{ccc} Hc & \xrightarrow{\alpha^c} & H'c \\ {\scriptstyle Hf}\downarrow & & \downarrow{\scriptstyle Hf'} \\ Hc' & \xrightarrow{\alpha^{c'}} & H'c' \end{array} \tag{2}$$

This diagram does not ask for naturality of a transformation $\alpha : H \to H'$, it is only expressed between the components α^c and $\alpha^{c'}$, and this only for morphism $f : c \to c'$. When running through all objects and all morphisms of C, the ingredients added to C in order to obtain $\mathcal{T}(H, H')$ constitute such a natural transformation $\alpha : H \to H'$.

There is a forgetful functor $U : \mathcal{T}(H, H') \to C$, and already from the previous description, it becomes clear that the natural transformations α from H to H' are in bijection with functors N from C to $\mathcal{T}(H, H')$ that satisfy $U \cdot N = 1_C$. This latter identity is between functors, and, if expressed naïvely, relies on equality of objects in a category. In particular, the type of such identities is not a proposition. The way out is to exploit that the target category is obtained through a displayed category. There is an elementary characterization of functors from a source category into the total category of a displayed category for which composition with the forgetful functor agrees with a given functor into the base of that target category, which in our case is just 1_C. The elementary concept here is the formalization of the notion of *section*, in particular section_disp. To show the above-mentioned correspondence, we thus concretely formalized a bijection between the natural transformations from H to H' and the sections with respect to C and the displayed category underlying $\mathcal{T}(H, H')$ (nat_trafo_to_section and section_to_nat_trafo).

It turns out that these observations can be smoothly carried over to an arbitrary bicategory for the elements in the target, instead of limiting them to functors and natural transformations. This gives a formalized bijection in form of nat_trafo_to_section_bicat and section_to_nat_trafo_bicat.

In the situation of parameterized distributivity, we are lacking the notion of displayed monoidal category and the suitable section characterization for strong monoidal functors from a given monoidal category into the total category of the displayed monoidal category whose composition with the forgetful functor yields a given strong monoidal functor from the source into the base. We leave the development of such a notion as an open problem. It is essentially not a mathematical problem but a formalization challenge—this time with the purpose of avoiding reasoning with equality between strong monoidal functors. In this paper, we can only offer one direction of the correspondence: given a parameterized distributivity, we construct a strong monoidal functor from the monoidal category $\mathcal{V}$ of parameters into a specially crafted monoidal category.

We use the general result above and instantiate C with $\mathcal{V}$ and H and H' with the source and target of δ and add monoidal structure to $\mathcal{T}(H, H')$. Recall that objects of the category $\mathcal{T}(H, H')$ contain natural transformations from $\mathcal{A}$ to $\mathcal{A}'$ and that morphisms of $\mathcal{T}(H, H')$ come with proof obligations for equations between such natural transformations. This allows us to precisely capture the diagrams of Fig. 5 in the definition of the objects of $\mathcal{T}(H, H')$ that are needed to turn it into a monoidal category: For the unit, we add to the unit I of $\mathcal{V}$ the natural transformation $(G \cdot \epsilon) \circ ((\epsilon')^{-1} \cdot G)$ to which δ_I is supposed to be equal. In order to get the tensor on objects (v, dv) and (w, dw) of $\mathcal{T}(H, H')$, we add to $v \otimes w$ the natural transformation that arises from taking in the pentagon diagram the path that is meant to correspond to $\delta_{v \otimes w}$, with δ_v and δ_w replaced by dv and dw, respectively. This extends to a tensor operation on $\mathcal{T}(H, H')$, but not without effort: we ask for the morphism of $\mathcal{T}(H, H')$ that corresponds to the tensor of two such morphisms, thus we have to prove an equality between two chains of compositions of natural transformations. And there are more morphisms of $\mathcal{T}(H, H')$ asked for in order to get left and right unitor and the associator. The formalization effort for these equalities turns out to be incommensurate with their proofs on paper, the main problem being that there is too much structure in those "concrete" functors and natural transformations, so that any attempt at simplifying the goals at hand makes them unreadably convoluted. The way out is a bicategorical generalization of the problem at hand, extending the bicategorical generalization of the correspondence for the case of plain categories we mentioned above. This does not mean that the formalization becomes an easy task, but there is still an adequation between the pencil-and-paper proof and the formalization. Moreover, a part of the proofs has been obtained without a prior pencil-and-paper proof, which is a major desideratum for the development of formalized mathematical results. Here, we briefly mention some of the more important elements of the formalization: montrafotargetbicat_cat is the target category, which is extended to a monoidal category montrafotargetbicat_moncat,

where the morphism part of the tensor rests on Lemma montrafotargetbicat_tensor_comp_aux with a proof of over 200 lines, and six more such lemmas for the unitors and the associator whose construction in total requires more than 1kloc. The laws themselves do not need much effort since the extra data for morphisms in $\mathcal{T}(H, H')$ consists of equalities, hence their equalities are trivial under our general assumption that the homsets of all categories in our formalization are sets in the sense of univalent foundations. The main result is the bicategorical generalization of the translation of parameterized distributivity into strong monoidal functors smf_from_param_distr_bicat_parts (the notion of parameterized distributivity is not yet properly generalized to bicategories), and the main result of this section is then the instantiation of the monoidal target category montrafotarget_moncat and of the translation itself smf_from_param_distr. It is in this sense that the strength laws express that G is "homomorphic". We leave as an open problem to formulate precisely, and prove, the full correspondence that strength laws are nothing but being "homomorphic" in the sense of our crafted monoidal target category.

5 Conclusions and Further Work

We have presented a mathematical framework for the specification and generation of simply-typed syntax, fully implemented and computer-checked in Coq, using the UniMath library of univalent mathematics. To this end, we generalized recent work by [13] on untyped syntax: (i) we generalized definitions and adapted the statements and proofs—this required very little effort, (ii) we extended the existing library on ω-cocontinuous functors, and (iii) we gave a new analysis of action-based strengths and embedded them into a bicategorical context.

It is difficult to provide exact numbers for what was involved in this work as it is a modification and extension of a major library of formalized mathematics. Indeed, some of the work was pure maintenance of the library and is difficult to measure accurately, e. g., the notion of isomorphism for monoidal categories had to be replaced which led to a major overhaul of various files related to isomorphisms more generally. The tool coqwc counts approximately 6,700 lines of statements and proofs of new files, not counting the improvements and additions to existing material.

Our work relies heavily on *universal properties*: both the syntax itself, as well as the substitution operation on it, are characterized through their universal properties. Since universal properties are conditions of unique existence, with computational content, the notions of contractible type and proposition provided by Univalent Foundations seem particularly well suited for discussing such work.

Our work benefited greatly from infrastructure provided by the UniMath library. Firstly, in Section 4 we heavily rely on the recent addition of bicategory theory to UniMath by Ahrens et al. [6]. A difficulty in bicategory theory is the sheer complexity of the involved notions; for instance a bicategory in [6] has 25 fields of increasing complexity. By working in a proof assistant one can be sure that no proof obligations have been overlooked. This resembles traditional metatheory of programming languages where many proofs proceed by induction on large inductive definitions, leading to a multitude of cases to consider. Secondly, the machinery of displayed categories [11] and displayed bicategories [6] helps building the complicated (bi)categories handled here.

A truly different approach to working with $\mathcal{C}^{\mathrm{S}}$ in the case of $\mathcal{C} = \mathrm{Set}$ would be to work with the equivalent slice category Set/S. The first version of the framework presented here was in fact implemented this way. However, it had many drawbacks: first of all it is less general, second, it was very clumsy to work with the constructed monad on the slice category, third, working in a specific category is often less convenient than in an abstract category (e. g., things often unfold further, quickly leading to unreadable goals).

We have focused on the mathematical construction of abstract syntax, and on the mathematical justification of recursion on that syntax. The terms of our syntax associated to a signature are equivalence classes, and thus not convenient to handle in practice. A suitable inductive datatype of terms as used, e. g. by Ahrens and Zsidó [14] would be more convenient to use. We anticipate that it is feasible to construct, generically, this datatype and an isomorphism between our type family of terms and the inductive family.

It is not immediately clear how to directly generalize our approach to dependently-typed syntax (going beyond the pre-syntax of Example 3.10). Voevodsky [62] set out a path for the construction of such syntax; it leads via pre-types and -terms in the form of monads as we construct them here, and considers C-systems (a. k. a. contextual categories [21]) built from such pre-syntax as the raw material out of which to carve out the desired syntax. The constructions presented here thus constitute an important stepping stone in Voevodsky's programme of formalizing the metatheory of type theory in Univalent Foundations.

Acknowledgments

We are grateful to Vladimir Voevodsky for many discussions on the subjects of multi-sorted term systems and the intended applications to type theory. We thank Peter LeFanu Lumsdaine for suggesting and formalizing an improvement to one of our proofs.

Anders Mörtberg was supported by the Swedish Research Council (SRC, Vetenskapsrådet) under Grant No. 2019-04545.

This material is based upon work supported by the National Science Foundation under Grant No. DMS-1128155 and CMU 1150129-338510.

This work has been partly funded by the CoqHoTT ERC Grant 637339.

References

[1] Jiří Adámek. 1974. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae* 15, 4 (1974), 589–602.

[2] Benedikt Ahrens. 2012. Extended Initiality for Typed Abstract Syntax. *Log. Methods Comput. Sci.* 8, 2 (2012). https://doi.org/10.2168/LMCS-8(2:1)2012

[3] Benedikt Ahrens. 2012. Initiality for Typed Syntax and Semantics. In *Logic, Language, Information and Computation - 19th International Workshop, WoLLIC 2012, Buenos Aires, Argentina, September 3-6, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7456)*, C.-H. Luke Ong and Ruy J. G. B. de Queiroz (Eds.). Springer, 127–141. https://doi.org/10.1007/978-3-642-32621-9_10

[4] Benedikt Ahrens. 2016. Modules over relative monads for syntax and semantics. *Math. Struct. Comput. Sci.* 26, 1 (2016), 3–37. https://doi.org/10.1017/S0960129514000103

[5] Benedikt Ahrens. 2019. Initial Semantics for Reduction Rules. *Log. Methods Comput. Sci.* 15, 1 (2019). https://doi.org/10.23638/LMCS-15(1:28)2019

[6] Benedikt Ahrens, Dan Frumin, Marco Maggesi, and Niels van der Weide. 2019. Bicategories in Univalent Foundations. In *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany (LIPIcs, Vol. 131)*, Herman Geuvers (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 5:1–5:17. https://doi.org/10.4230/LIPIcs.FSCD.2019.5

[7] Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. 2019. Modular Specification of Monads Through Higher-Order Presentations. In *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany (LIPIcs, Vol. 131)*, Herman Geuvers (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 6:1–6:19. https://doi.org/10.4230/LIPIcs.FSCD.2019.6

[8] Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. 2020. Reduction monads and their signatures. *Proc. ACM Program. Lang.* 4, POPL (2020), 31:1–31:29. https://doi.org/10.1145/3371099

[9] Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. 2021. Presentable signatures and initial semantics. *Log. Methods Comput. Sci.* 17, 2 (2021). https://doi.org/10.23638/LMCS-17(2:17)2021

[10] Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. 2015. Univalent categories and the Rezk completion. *Math. Struct. in Comp. Science* 25 (2015), 1010–1039. https://doi.org/10.1017/S0960129514000486 arXiv:1303.0584

[11] Benedikt Ahrens and Peter LeFanu Lumsdaine. 2019. Displayed Categories. *Log. Methods Comput. Sci.* 15, 1 (2019). https://doi.org/10.23638/LMCS-15(1:20)2019

[12] Benedikt Ahrens and Ralph Matthes. 2015. Heterogeneous Substitution Systems Revisited. In *21st International Conference on Types for Proofs and Programs, TYPES 2015, May 18-21, 2015, Tallinn, Estonia (LIPIcs, Vol. 69)*, Tarmo Uustalu (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2:1–2:23. https://doi.org/10.4230/LIPIcs.TYPES.2015.2

[13] Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. 2019. From Signatures to Monads in UniMath. *J. Autom. Reason.* 63, 2 (2019), 285–318. https://doi.org/10.1007/s10817-018-9474-4

[14] Benedikt Ahrens and Julianna Zsidó. 2011. Initial Semantics for higher-order typed syntax in Coq. *J. Formaliz. Reason.* 4, 1 (2011), 25–69. https://doi.org/10.6092/issn.1972-5787/2066

[15] Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. 2018. A type and scope safe universe of syntaxes with binding: their semantics and proofs. *Proc. ACM Program. Lang.* 2, ICFP (2018), 90:1–90:30. https://doi.org/10.1145/3236785

[16] Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. 2020. A Type and Scope Safe Universe of Syntaxes with Binding: Their Semantics and Proofs. *CoRR* abs/2001.11001 (2020). arXiv:2001.11001 https://arxiv.org/abs/2001.11001

[17] Thorsten Altenkirch, Neil Ghani, Peter G. Hancock, Conor McBride, and Peter Morris. 2015. Indexed containers. *J. Funct. Program.* 25 (2015). https://doi.org/10.1017/S095679681500009X

[18] Thorsten Altenkirch and Bernhard Reus. 1999. Monadic Presentations of Lambda Terms Using Generalized Inductive Types. In *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1683)*, Jörg Flum and Mario Rodríguez-Artalejo (Eds.). Springer, 453–468. https://doi.org/10.1007/3-540-48168-0_32

[19] Françoise Bellegarde and James Hook. 1994. Substitution: A Formal Methods Case Study Using Monads and Transformations. *Sci. Comput. Program.* 23, 2-3 (1994), 287–311. https://doi.org/10.1016/0167-6423(94)00022-0

[20] Richard S. Bird and Ross Paterson. 1999. De Bruijn Notation as a Nested Datatype. *J. Funct. Program.* 9, 1 (1999), 77–91. http://journals.cambridge.org/action/displayAbstract?aid=44239

[21] John Cartmell. 1986. Generalised algebraic theories and contextual categories. *Ann. Pure Appl. Logic* 32 (1986), 209–243. https://doi.org/10.1016/0168-0072(86)90053-9

[22] James Chapman, Pierre-Évariste Dagand, Conor McBride, and Peter Morris. 2010. The gentle art of levitation. In *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010, Baltimore, Maryland, USA, September 27-29, 2010*, Paul Hudak and Stephanie Weirich (Eds.). ACM, 3–14. https://doi.org/10.1145/1863543.1863547

[23] Pierre-Évariste Dagand and Conor McBride. 2013. A Categorical Treatment of Ornaments. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*. IEEE Computer Society, 530–539. https://doi.org/10.1109/LICS.2013.60

[24] Marcelo Fiore and Dmitrij Szamozvancev. 2021. Formal Metatheory of Second-Order Abstract Syntax. *To be published in Principles of Programming Languages (POPL) 2022* (2021). https://www.repository.cam.ac.uk/handle/1810/330658

[25] Marcelo P. Fiore. 2002. Semantic analysis of normalisation by evaluation for typed lambda calculus. In *Proceedings of the 4th international ACM SIGPLAN conference on Principles and practice of declarative programming, October 6-8, 2002, Pittsburgh, PA, USA (Affiliated with PLI 2002)*. ACM, 26–37. https://doi.org/10.1145/571157.571161

[26] Marcelo P. Fiore. 2008. Second-Order and Dependently-Sorted Abstract Syntax. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*. IEEE Computer Society, 57–68. https://doi.org/10.1109/LICS.2008.38

[27] Marcelo P. Fiore. 2012. Discrete Generalised Polynomial Functors - (Extended Abstract). In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 7392)*, Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer (Eds.). Springer, 214–226. https://doi.org/10.1007/978-3-642-31585-5_22

[28] Marcelo P. Fiore and Makoto Hamana. 2013. Multiversal Polymorphic Algebraic Theories: Syntax, Semantics, Translations, and Equational Logic. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*. IEEE Computer Society, 520–529. https://doi.org/10.1109/LICS.2013.59

[29] Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. 1999. Abstract Syntax and Variable Binding. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*. IEEE Computer Society, 193–202. https://doi.org/10.1109/LICS.1999.782615

[30] Murdoch Gabbay and Andrew M. Pitts. 1999. A New Approach to Abstract Syntax Involving Binders. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*. IEEE Computer Society, 214–224. https://doi.org/10.1109/LICS.1999.782617

[31] Nicola Gambino and Martin Hyland. 2003. Wellfounded Trees and Dependent Polynomial Functors. In *Types for Proofs and Programs, International Workshop, TYPES 2003, Torino, Italy, April 30 - May 4, 2003, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 3085)*, Stefano Berardi, Mario Coppo, and Ferruccio Damiani (Eds.). Springer, 210–225. https://doi.org/10.1007/978-3-540-24849-1_14

[32] Makoto Hamana. 2011. Polymorphic Abstract Syntax via Grothendieck Construction. In *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6604)*, Martin Hofmann (Ed.). Springer, 381–395. https://doi.org/10.1007/978-3-642-19805-2_26

[33] Makoto Hamana and Marcelo P. Fiore. 2011. A foundation for GADTs and inductive families: dependent polynomial functor approach. In *Proceedings of the seventh ACM SIGPLAN workshop on Generic programming, WGP@ICFP 2011, Tokyo, Japan, September 19-21, 2011*, Jaakko Järvi and Shin-Cheng Mu (Eds.). ACM, 59–70. https://doi.org/10.1145/2036918.2036927

[34] André Hirschowitz, Tom Hirschowitz, and Ambroise Lafont. 2020. Modules over Monads and Operational Semantics. In *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference) (LIPIcs, Vol. 167)*, Zena M. Ariola (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 12:1–12:23. https://doi.org/10.4230/LIPIcs.FSCD.2020.12

[35] André Hirschowitz, Tom Hirschowitz, and Ambroise Lafont. 2020. Modules over Monads and Operational Semantics. (2020). arXiv:2012.06530v1

[36] André Hirschowitz and Marco Maggesi. 2007. Modules over Monads and Linearity. In *WoLLIC (Lecture Notes in Computer Science, Vol. 4576)*, Daniel Leivant and Ruy J. G. B. de Queiroz (Eds.). Springer, 218–237.

[37] André Hirschowitz and Marco Maggesi. 2010. Modules over monads and initial semantics. *Inf. Comput.* 208, 5 (2010), 545–564.

[38] Martin Hofmann. 1999. Semantical Analysis of Higher-Order Abstract Syntax. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*. IEEE Computer Society, 204–213. https://doi.org/10.1109/LICS.1999.782616

[39] Chung-Kil Hur. 2010. *Categorical equational systems : algebraic models and equational reasoning*. Ph. D. Dissertation. University of Cambridge, UK. http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.608664

[40] George Janelidze and Gregory Maxwell Kelly. 2001. A Note on Actions of a Monoidal Category. *Theory and Applications of Categories* 9, 4 (2001), 61–91. Online available at http://www.tac.mta.ca/tac/volumes/9/n4/9-04abs.html.

[41] Gyesik Lee, Bruno C. d. S. Oliveira, Sungkeun Cho, and Kwangkeun Yi. 2012. GMeta: A Generic Formal Metatheory Framework for First-Order Representations. In *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7211)*, Helmut Seidl (Ed.). Springer, 436–455. https://doi.org/10.1007/978-3-642-28869-2_22

[42] Andres Löh and José Pedro Magalhães. 2011. Generic programming with indexed functors. In *Proceedings of the seventh ACM SIGPLAN workshop on Generic programming, WGP@ICFP 2011, Tokyo, Japan, September 19-21, 2011*, Jaakko Järvi and Shin-Cheng Mu (Eds.). ACM, 1–12. https://doi.org/10.1145/2036918.2036920

[43] Saunders Mac Lane. 1998. *Categories for the Working Mathematician* (second ed.). Graduate Texts in Mathematics, Vol. 5. Springer-Verlag, New York. xii+314 pages.

[44] Ola Mahmoud. 2011. *Second-order algebraic theories*. Ph. D. Dissertation. University of Cambridge, UK. http://www.dspace.cam.ac.uk/handle/1810/241035

[45] Per Martin-Löf. 1984. *Intuitionistic type theory*. Studies in proof theory, Vol. 1. Bibliopolis.

[46] Ralph Matthes and Tarmo Uustalu. 2004. Substitution in non-wellfounded syntax with variable binding. *Theoretical Computer Science* 327, 1-2 (2004), 155–174. https://doi.org/10.1016/j.tcs.2004.07.025

[47] Nax Paul Mendler. 1991. Inductive types and type constraints in the second-order lambda calculus. *Ann. Pure Appl. Logic* 51, 1-2 (1991), 159–172. https://doi.org/10.1016/0168-0072(91)90069-X

[48] Marino Miculan and Ivan Scagnetto. 2003. A framework for typed HOAS and semantics. In *Proceedings of the 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 27-29 August 2003, Uppsala, Sweden*. ACM, 184–194. https://doi.org/10.1145/888251.888269

[49] Bodo Pareigis. 1977. Non-additive Ring and Module Theory II. C-categories, C-functors, and C-morphisms. *Publicationes Mathematicae Debrecen* 24 (1977), 351–361.

[50] G.D. Plotkin. 1977. LCF considered as a programming language. *Theoretical Computer Science* 5, 3 (1977), 223–255. https://doi.org/10.1016/0304-3975(77)90044-5

[51] Emmanuel Polonowski. 2013. Automatically Generated Infrastructure for De Bruijn Syntaxes. In *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7998)*, Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie (Eds.). Springer, 402–417. https://doi.org/10.1007/978-3-642-39634-2_29

[52] Nicolas Pouillard. 2012. *Namely, Painless: A unifying approach to safe programming with first-order syntax with binders*. Theses. Université Paris-Diderot - Paris VII. https://tel.archives-ouvertes.fr/tel-00759059

[53] Nicolas Pouillard and François Pottier. 2012. A unified treatment of syntax with binders. *J. Funct. Program.* 22, 4-5 (2012), 614–704. https://doi.org/10.1017/S0956796812000251

[54] Steven Schäfer, Tobias Tebbi, and Gert Smolka. 2015. Autosubst: Reasoning with de Bruijn Terms and Parallel Substitutions. In *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9236)*, Christian Urban and Xingyuan Zhang (Eds.). Springer, 359–374. https://doi.org/10.1007/978-3-319-22102-1_24

[55] Peter Sewell, Francesco Zappa Nardelli, Scott Owens, Gilles Peskine, Thomas Ridge, Susmit Sarkar, and Rok Strnisa. 2010. Ott: Effective tool support for the working semanticist. *J. Funct. Program.* 20, 1 (2010), 71–122. https://doi.org/10.1017/S0956796809990293

[56] Kathrin Stark, Steven Schäfer, and Jonas Kaiser. 2019. Autosubst 2: reasoning with multi-sorted de Bruijn terms and vector substitutions. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, Assia Mahboubi and Magnus O. Myreen (Eds.). ACM, 166–180. https://doi.org/10.1145/3293880.3294101

[57] Thomas Streicher. 1991. *Semantics of type theory - correctness, completeness and independence results*. Birkhäuser.

[58] Miki Tanaka and John Power. 2005. A unified category-theoretic formulation of typed binding signatures. In *Proceedings of the 3rd ACM SIGPLAN workshop on Mechanized reasoning about languages with variable binding* (Tallinn, Estonia) *(MERLIN '05)*. ACM, New York, NY, USA, 13–24. https://doi.org/10.1145/1088454.1088457

[59] The Coq Development Team. 2021. The Coq Proof Assistant, version 8.13.0. https://doi.org/10.5281/zenodo.4501022

[60] The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book, Institute for Advanced Study.

[61] Vladimir Voevodsky. 2015. An experimental library of formalized Mathematics based on the univalent foundations. *Math. Struct. Comput. Sci.*

25, 5 (2015), 1278–1294. https://doi.org/10.1017/S0960129514000577

[62] Vladimir Voevodsky. 2016. C-system of a module over a Jf-relative monad. (2016). https://arxiv.org/abs/1602.00352.

[63] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. 2021. UniMath — a computer-checked library of univalent mathematics. Available at http://unimath.github.io/UniMath/ .

[64] Stephanie Weirich and Brian Aydemir. 2010. *LNgen: Tool Support for Locally Nameless Representations*. Technical Report. https://repository.upenn.edu/cis_reports/933/

[65] Julianna Zsidó. 2010. *Typed Abstract Syntax*. Ph. D. Dissertation. University of Nice, France. http://tel.archives-ouvertes.fr/tel-00535944/.